# UNIX SOCKET - NETWORK BYTE ORDERS

 Copyright © tutorialspoint.com

Unfortunately, not all computers store the bytes that comprise a multibyte value in the same order. Consider a 16-bit internet that is made up of 2 bytes. There are two ways to store this value.

- **Little Endian**: In this scheme, low-order byte is stored on the starting address $A$ and high-order byte is stored on the next address $A + 1$.
- **Big Endian**: In this scheme, high-order byte is stored on the starting address $A$ and low-order byte is stored on the next address $A + 1$.

To allow machines with different byte order conventions communicate with each other, the Internet protocols specify a canonical byte order convention for data transmitted over the network. This is known as Network Byte Order.

While establishing an Internet socket connection, you must make sure that the data in the sin_port and sin_addr members of the sockaddr_in structure are represented in Network Byte Order.

## Byte Ordering Functions

Routines for converting data between a host's internal representation and Network Byte Order are as follows:

| Function | Description |
|---|---|
| htons | Host to Network Short |
| htonl | Host to Network Long |
| ntohl | Network to Host Long |
| ntohs | Network to Host Short |

Listed below are some more detail about these functions:

- **unsigned short htons*unsignedshorthostshort***

  This function converts 16-bit $2 - byte$ quantities from host byte order to network byte order.

- **unsigned long htonl*unsignedlonghostlong***

  This function converts 32-bit $4 - byte$ quantities from host byte order to network byte order.

- **unsigned short ntohs*unsignedshortnetshort***

  This function converts 16-bit $2 - byte$ quantities from network byte order to host byte order.

- **unsigned long ntohl*unsignedlongnetlong***

  This function converts 32-bit quantities from network byte order to host byte order.

These functions are macros and result in the insertion of conversion source code into the calling program. On little-endian machines, the code will change the values around to network byte order. On big-endian machines, no code is inserted since none is needed; the functions are defined as null.

## Program to Determine Host Byte Order

Keep the following code in a file *byteorder.c* and then compile it and run it over your machine.

In this example, we store the two-byte value 0x0102 in the short integer and then look at the two

consecutive bytes, c[0] $theaddressA$ and c[1] $theaddressA + 1$ to determine the byte order.

```
#include <stdio.h>

int main(int argc, char **argv)
{
   union {
      short s;
      char c[sizeof(short)];
   }un;
   un.s = 0x0102;

   if (sizeof(short) == 2) {
      if (un.c[0] == 1 && un.c[1] == 2)
         printf("big-endian\n");

      else if (un.c[0] == 2 && un.c[1] == 1)
         printf("little-endian\n");

      else
         printf("unknown\n");
      }

      else {
         printf("sizeof(short) = %d\n", sizeof(short));
      }
   exit(0);
}
```

An output generated by this program on a Pentium machine is as follows:

```
$> gcc byteorder.c
$> ./a.out
little-endian
$>
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js